

# Interior Point Methods applied to (n-1) secure and stochastic Optimal Power Flow formuatuons

Optimization in Energy Systems, Snowbird, UT

Naiyuan Chiang<sup>1</sup>    Jacek Gondzio<sup>1</sup>    Andreas Grothey<sup>1</sup>  
                                  Chris Dent<sup>2</sup>

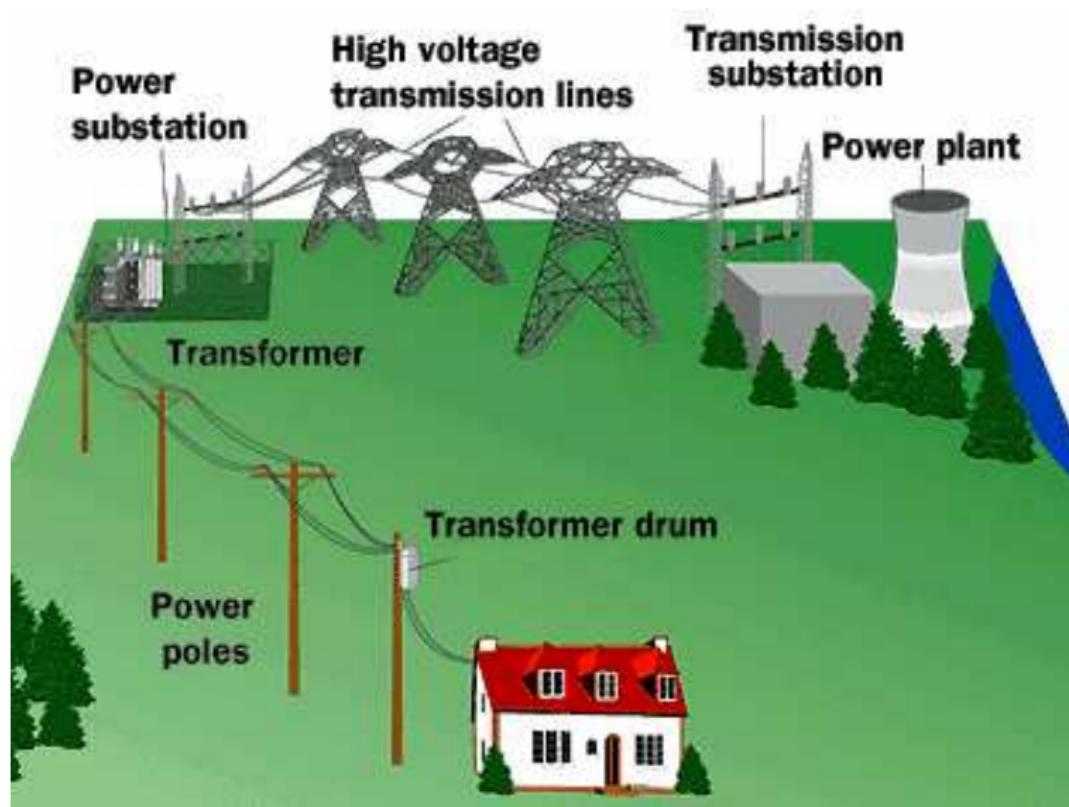
<sup>1</sup>School of Mathematics  
University of Edinburgh

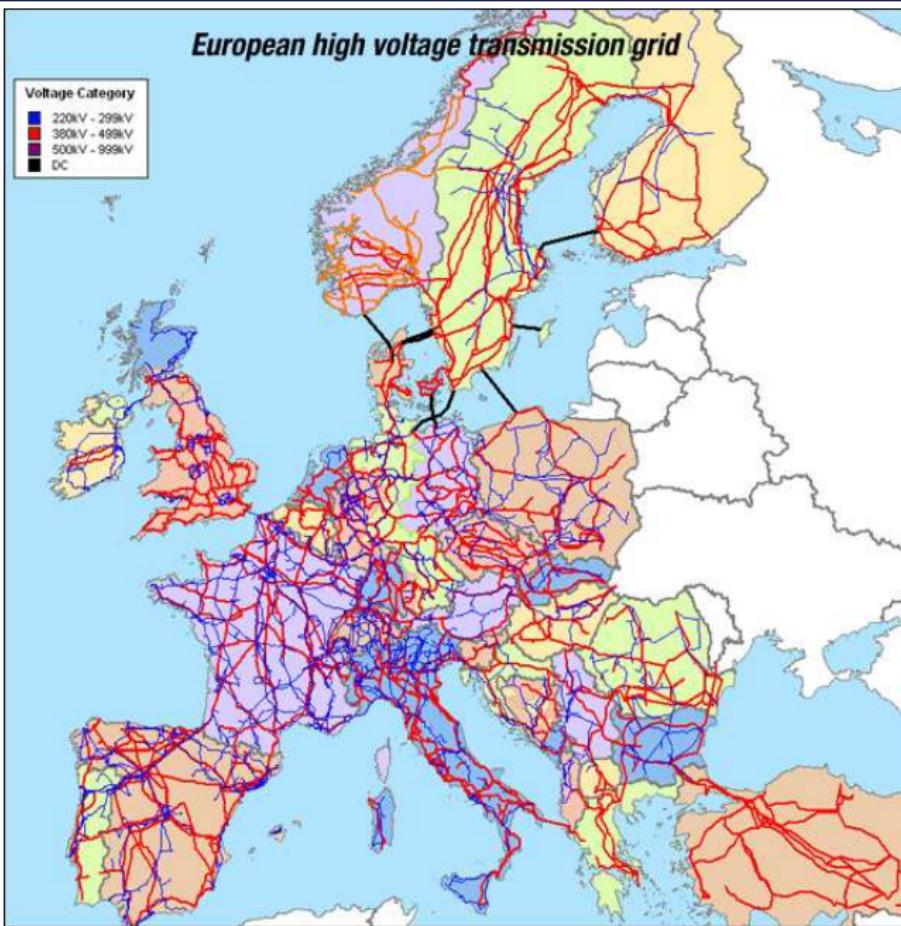
<sup>2</sup>Electrical Engineering  
University of Durham

August 4, 2010

- 1 AC and DC Optimal Power Flow Models
- 2 Robust OPF Formulations
- 3 Interior Point Methods
- 4 Contingency Generation
- 5 Modelling







# Generic AC OPF Model

## Sets

- $b \in \mathcal{B}$  Buses
- $g \in \mathcal{G}$  Generators
- $l \in \mathcal{L}$  Lines

## Parameters

$\alpha_l, \beta_l$	conductance and susceptance of line $l$
$\beta_b$	susceptance of power source at bus $b$
$d_b^P, d_b^Q$	real and reactive power demand at bus $b$
$\bar{f}_l$	flow limit for line $l$

## Variables

$V_b$	Voltage level at bus $b$
$\delta_b$	Phase angle at bus $b$
$P_g, Q_g$	Real and reactive power input at generator $g$
$F_{(i,j)}^P, F_{(i,j)}^Q$	Real and reactive power flow on line $l = (i,j)$

# Generic AC OPF Model

## Constraints

- Kirchhoff Voltage Law (KVL)

$$F_{(i,j)}^P = V_i[\alpha_I(V_i - V_j \cos(\delta_i - \delta_j)) + \beta_I V_j \sin(\delta_i - \delta_j)]$$

$$F_{(i,j)}^Q = V_i[\beta_I(V_j \sin(\delta_i - \delta_j) - V_i) + \alpha_I V_j \cos(\delta_i - \delta_j)]$$

- Kirchhoff Current Law (KCL)

$$\sum_{g|o_g=b} P_g = \sum_{(b,i) \in L} F_{(b,i)}^P + d_b^P, \quad \forall b \in \mathcal{B},$$

$$\sum_{g|o_g=b} Q_g - \beta_b V_b^2 = \sum_{(b,i) \in L} F_{(b,i)}^Q + d_b^Q, \quad \forall b \in \mathcal{B}$$

- Line Flow Limits at both ends of each line

$$(F_{(i,j)}^P)^2 + (F_{(i,j)}^Q)^2 \leq \bar{f}_l^2$$

$$(F_{(j,i)}^P)^2 + (F_{(j,i)}^Q)^2 \leq \bar{f}_l^2$$

# Structure of AC OPF problem

Can use KVL to eliminate  $F_{(i,j)}^P, F_{(i,j)}^Q \Rightarrow$  left with  $\Delta, V, P/Q$

- Flow limits at *to* and *from* nodes:

$$\begin{aligned} h_f(\Delta, V) &\leq 0 \\ h_t(\Delta, V) &\leq 0 \end{aligned}$$

- Kirchhoff Current Law

$$\begin{aligned} P &= g^P(\Delta, V) \\ Q &= g^Q(\Delta, V) \end{aligned}$$

$$\Delta = (\delta_1, \dots, \delta_{|\mathcal{B}|}), V = (V_1, \dots, V_{|\mathcal{B}|}), P = (P_1, \dots, P_{|\mathcal{G}|}), Q = (Q_1, \dots, Q_{|\mathcal{G}|})$$

⇒ The AC OPF is a nonlinear programming problem

# Structure of DC OPF problem

Model is simplified under the following assumptions:

- The voltage level at all buses is the same:  $V$ .
- The resistance of each line is small (compared to reactance):  
 $\Rightarrow \alpha_I = 0, \beta_I = -1/r_I, r_I$ : resistance of line  $I$
- The phase angle difference between each two buses is small  
 $\Rightarrow \sin(\delta_1 - \delta_2) \approx \delta_1 - \delta_2, \cos(\delta_1 - \delta_2) \approx 1, \Rightarrow F_{(i,j)}^Q = 0$ .

## DC-OPF model

- Kirchhoff Current Law

$$\sum_{g|o_g=b} P_g = \sum_{(b,i) \in \mathcal{L}} F_{(b,i)}^P + d_b^P, \quad \forall b \in \mathcal{B}$$

- Kirchhoff Voltage Law

$$F_I^P = -\frac{V^2}{r_I} \sum_{b \in \mathcal{B}} a_{bI} \delta_b, \quad \forall I \in \mathcal{L}$$

- Line Flow Limits:  $-\bar{f}_I \leq F_I^P \leq \bar{f}_I, \quad \forall I \in \mathcal{L}$

# Structure of DC OPF problem

Given

- bus/generator incidence matrix  $J \in \mathbb{R}^{|\mathcal{B}| \times |\mathcal{G}|}$
- node/arc incidence matrix  $A \in \mathbb{R}^{|\mathcal{B}| \times |\mathcal{L}|}$
- $R = \text{diag}(-V^2/r_1, \dots, -V^2/r_{|\mathcal{L}|})$

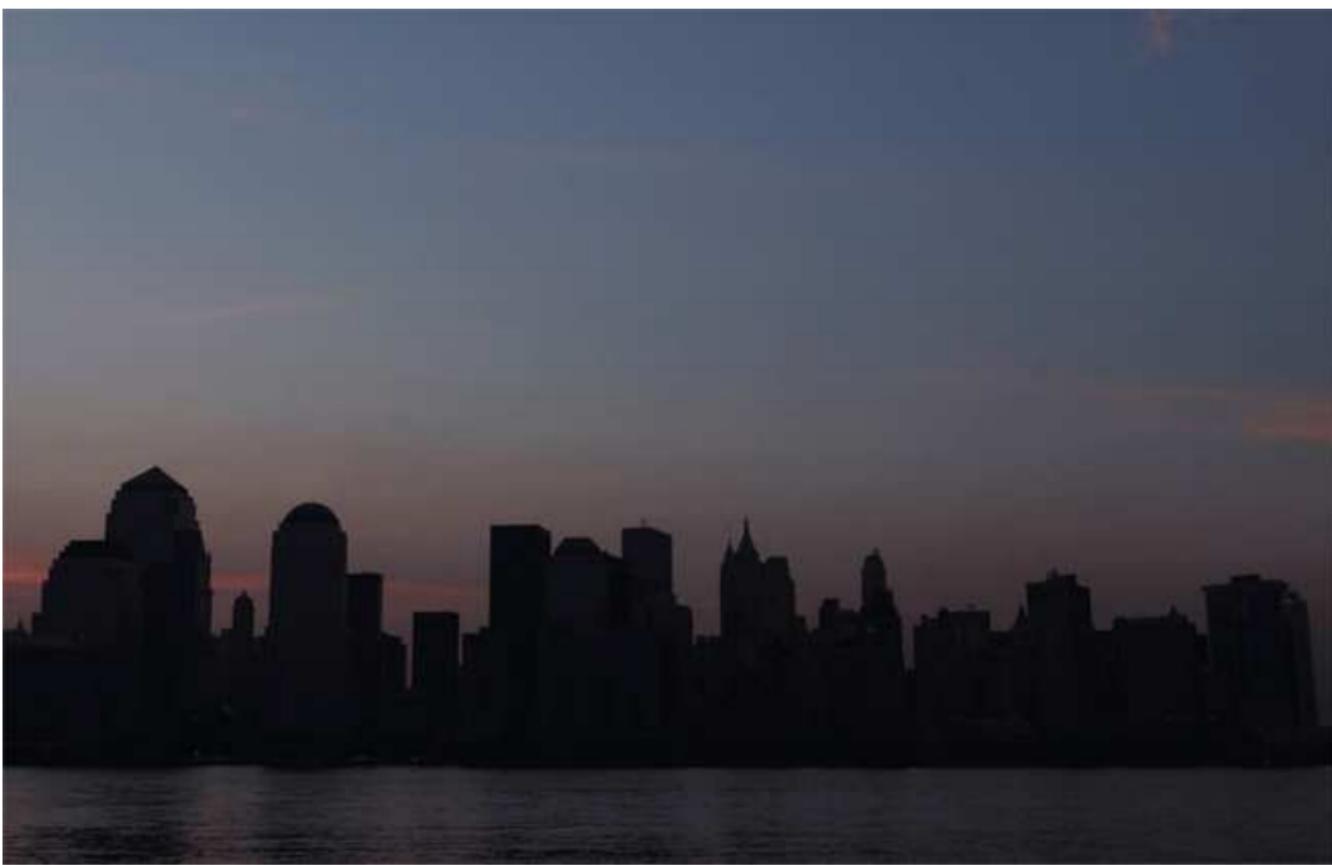
the DC-OPF problem can be written as

## DC-OPF

$$\begin{array}{ll}\min & c^\top P \\ \text{s.t.} & RF + A^\top \Delta = 0 \\ & AF - JP = -d\end{array}$$

⇒ DC OPF is a linear programming problem

Question: When is this a good/adequate approximation?





# Robust OPF formulations

## (n-1)-secure

Classical approach: Network should survive the failure of any one bus or line (possibly after limited corrective actions) **without line-overloads.**

## Stochastic demand/generation

Network should have flexibility to cope with stochastically changing demand/generation (after corrective actions) **without line overloads**

## Robust demand/generation

Network should be able to cope with the worst case demand/generation scenario within a given confidence set.

# (n-1) secure OPF

## Setup

- Contingency scenarios  $c \in \mathcal{C}$ , each has its network matrix  $A_c$ ,
- Real generation  $P$  and Voltage  $V_g$  same for all contingencies,
- Each contingency has its flow, voltage, phase angle and reactive generation:  $F_c^{P/Q}, V_c, \Delta_c, Q_c$ ,
- Possible modification of generator output  $\Delta P_c$  in each contingency scenario.
- Seek a generator setting that does not create line overloads for any contingency

## DC SCOPF

$$\begin{aligned} \min \quad & c^T P \\ \text{s.t.} \quad & RF_c + A_c^T \Delta_c = 0, \quad \forall c \in \mathcal{C} \\ & A_c F_c = JP - d, \quad \forall c \in \mathcal{C} \end{aligned}$$

# Structure of (n-1) secure DC OPF

$$\begin{array}{llll} \min_{P, \Delta, F} & & c^\top P & \\ \text{s.t.} & RF_1 + A_1^\top \Delta_1 & = 0 \\ & A_1 F_1 & -JP & = d \\ & & \ddots & \vdots \\ & RF_{|\mathcal{C}|} + A_{|\mathcal{C}|}^\top \Delta_{|\mathcal{C}|} & = 0 \\ & A_{|\mathcal{C}|} F_{|\mathcal{C}|} & -JP & = d \end{array}$$

# Structure of (n-1) secure DC OPF

$$\begin{aligned} \min_{P, \Delta, F} \\ \text{s.t.} \end{aligned}$$

$$\boxed{\begin{array}{l} RF_1 \\ A_1^T \end{array} \boxed{W_1^+} \begin{array}{l} A_1^\top \Delta_1 \\ A_1 F_1 \end{array}}$$

$\vdots$   
 $\vdots$

$$\boxed{\begin{array}{l} RF_{|c|} \\ A_{|c|}^T \end{array} \boxed{W_{|c|}^+} \begin{array}{l} A_{|c|}^\top \Delta_{|c|} \\ A_{|c|} F_{|c|} \end{array}}$$

$$\boxed{\begin{array}{l} c^\top P \\ -T_1 \\ \vdots \\ -T_{|c|} \end{array}} = \begin{array}{l} 0 \\ d \\ \vdots \\ 0 \\ d \end{array}$$

- Bordered block-diagonal matrix.

# Structure of (n-1) secure AC OPF

## Structure of Jacobian

$$\begin{bmatrix} u_1 & Q_1 & \cdots & u_{|\mathcal{C}|} & Q_{|\mathcal{C}|} & V_g & P \\ \hline H_u^{f/t} & & & & H_{V_g}^{f/t} & & \\ G_u^Q & -J & & & G_{V_g}^Q & & \\ G_u^P & & & & G_{V_g}^P & -J & \\ & \ddots & & & \vdots & \vdots & \\ & & H_u^{f/t} & & H_{V_g}^{f/t} & & \\ & & G_u^Q & -J & G_{V_g}^Q & & \\ & & G_u^P & & G_{V_g}^P & -J & \end{bmatrix}$$

$$u = (\Delta, V_b), H_u^{f/t} = \frac{\partial h^{f/t}}{\partial u}, H_{V_g}^{f/t} = \frac{\partial h^{f/t}}{\partial V_g}, G_u^Q = \frac{\partial g^P}{\partial u}, G_u^P = \frac{\partial g^Q}{\partial u}, \dots$$

# Structure of (n-1) secure AC OPF

## Structure of Jacobian



$$u = (\Delta, V_b), H_u^{f/t} = \frac{\partial h^{f/t}}{\partial u}, H_{V_g}^{f/t} = \frac{\partial h^{f/t}}{\partial V_g}, G_u^Q = \frac{\partial g^P}{\partial u}, G_u^P = \frac{\partial g^Q}{\partial u}, \dots$$



- UK target of  $\approx 30\%$  wind generation by 2020
- Licenses for development of 32 GW offshore by 2020 announced earlier this year
- Transmission is a major problem

# OPF with stochastic demand/generation

## Stochastic Programming Setup

- Decide on initial (real) generation  $P, V_g$ ,
- Several demand scenarios  $d_s^P, d_s^Q$ , each with probability  $\pi_s$ ,
- Uncertain (wind) generation modelled as *negative demand*.
- Each scenario has its Flow, Voltage, Phase, reactive generation:  $F_s^{P/Q}, V_s, \Delta_s, Q_s$ .
- After demand/wind is observed generator output  $\Delta P_s$  may be adjusted (within bounds) in each scenario.

$$P, V_g \longrightarrow d_s^{P/q} \longrightarrow F_s^{P/Q}, V_s, \Delta_s, Q_s, \Delta P_s$$

## DC OPF with stochastic demand/generation

$$\begin{aligned} \min \quad & c^\top P + \sum_s \pi_s c_2^\top \Delta P_s \\ \text{s.t.} \quad & RF_s + A^\top \Delta_s = 0, \quad \forall s \in \mathcal{S} \\ & AF_s - J\Delta P_s = JP - d_s, \quad \forall s \in \mathcal{S} \end{aligned}$$

# Structure of DC OPF with stochastic demand

$$\begin{array}{llll} \min_{P, \Delta, F} & \pi_1 c_2^\top \Delta P_1 + \cdots & + \pi_n c_2^\top \Delta P_n + c^\top P \\ \text{s.t.} & RF_1 + A^\top \Delta_1 & = 0 \\ & A_1 F_1 - J \Delta P_1 & - JP = d_1 \\ & & \ddots & \vdots = \vdots \\ & RF_n + A^\top \Delta_n & = 0 \\ & A_n F_n - J \Delta P_n - JP = d_n \end{array}$$

# Structure of DC OPF with stochastic demand

$$\begin{array}{ll}
 \min_{P, \Delta, F} & \pi_1 c_2^\top \Delta P_1 + \cdots + \pi_n c_2^\top \Delta P_n + c^\top P \\
 \text{s.t.} & \boxed{RF_1 + A_1^\top \Delta_1} \quad \boxed{W_1 - J\Delta P_1} = 0 \\
 & \boxed{A_1 F_1} \\
 & \vdots \\
 & \boxed{RF_n + A_n^\top \Delta_n} \quad \boxed{W_n - J\Delta P_n} = 0 \\
 & \boxed{A_n F_n} \quad \boxed{-JP} = d_1 \\
 & \vdots \\
 & \boxed{T_1} = 0 \\
 & \vdots \\
 & \boxed{T_n} = 0 \\
 & \boxed{-JR} = d_n
 \end{array}$$

- Bordered block-diagonal matrix.

# Risk modelling for OPF

Stochastic Programming facilitates Risk Modelling

## Possible model extensions

- Optimize expected cost of initial and recourse decisions.
- Bound risk exposure (measured by variance of recourse cost)
- Bound VaR/CVaR (risk exposure in worst p% of events).
- Require to outperform a given benchmark  
(→ Stochastic Dominance)

## Combined model

- The (n-1) secure and stochastic demand/generation model are of the same structure.
- Can be combined into a model incorporating both contingency and demand/generation scenarios.
- Can assign probabilities to contingency scenarios (to allow risk modelling)  
*Lamadrid et al '08: SuperOPF framework*

# Interior Point Methods (for LP)

## Linear Program

$$\begin{array}{ll} \min c^T x & \text{s.t. } Ax = b \\ & x \geq 0 \end{array} \quad (\text{LP})$$

## KKT Conditions

$$\begin{array}{lll} c - A^T \lambda - s & = & 0 \\ Ax & = & b \\ XSe & = & 0 \\ x, s & \geq & 0 \end{array} \quad (\text{KKT})$$

$X = \text{diag}(x), S = \text{diag}(s)$

# Interior Point Methods (for LP)

## Barrier Problem

$$\begin{aligned} \min c^\top x - \mu \sum \ln x_i & \quad \text{s.t.} \quad Ax = b \\ & \quad x \geq 0 \end{aligned} \quad (\text{LP}_\mu)$$

## KKT Conditions

$$\begin{aligned} c - A^\top \lambda - s &= 0 \\ Ax &= b \\ XSe &= \mu e \\ x, s &\geq 0 \end{aligned} \quad (\text{KKT}_\mu)$$

- Introduce logarithmic barriers for  $x \geq 0$

# Interior Point Methods (for LP)

## Barrier Problem

$$\min c^\top x - \mu \sum \ln x_i \quad \text{s.t.} \quad \begin{matrix} Ax = b \\ x \geq 0 \end{matrix} \quad (\text{LP}_\mu)$$

## KKT Conditions

$$\begin{aligned} c - A^\top \lambda - s &= 0 \\ Ax &= b \\ XSe &= \mu e \\ x, s &\geq 0 \end{aligned} \quad (\text{KKT}_\mu)$$

- Introduce logarithmic barriers for  $x \geq 0$
- $(\text{LP}_\mu)$  is strictly convex
- System  $(\text{KKT}_\mu)$  can be solved per Newton-Method

# Interior Point Methods (for LP)

## Barrier Problem

$$\begin{aligned} \min \quad & c^\top x - \mu \sum \ln x_i \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \quad (\text{LP}_\mu)$$

## KKT Conditions

$$\begin{aligned} c - A^\top \lambda - s &= 0 \\ Ax &= b \\ XSe &= \mu e \\ x, s &\geq 0 \end{aligned} \quad (\text{KKT}_\mu)$$

- Introduce logarithmic barriers for  $x \geq 0$
- $(\text{LP}_\mu)$  is strictly convex
- System  $(\text{KKT}_\mu)$  can be solved per Newton-Method
- For  $\mu \rightarrow 0$  solution of  $(\text{LP}_\mu)$  converges to solution of  $(\text{LP})$

# Interior Point Methods (for LP)

## KKT conditions

$$\begin{aligned} c - A^\top \lambda - s &= 0 \\ Ax &= b \\ XSe &= \mu e \\ x, s &\geq 0 \end{aligned} \tag{KKT}_\mu$$

# Interior Point Methods (for LP)

## KKT conditions

$$\begin{aligned}
 c - A^\top \lambda - s &= 0 \\
 Ax &= b \\
 XSe &= \mu e \\
 x, s &\geq 0
 \end{aligned} \tag{KKT}_\mu$$

## Central Path

The set of all solutions to  $(\text{KKT}_\mu)$  for all  $\mu > 0$ .

Central Path joins the **analytical center** (for  $\mu = \infty$ ) with the LP solution (for  $\mu = 0$ ).

## Neighbourhoods (of the central path)

$$\mathcal{N}_2(\theta) := \{(x, \lambda, s) \in \mathcal{F}^0 : \|XSe - \mu e\|_2 \leq \theta \mu\}$$

$$\mathcal{N}_{-\infty}(\gamma) := \{(x, \lambda, s) \in \mathcal{F}^0 : x_i s_i \geq \gamma \mu\}$$

where  $\mathcal{F}^0 := \{(x, \lambda, s) : c - A^\top \lambda - s = 0, Ax = b, x > 0, s > 0\}$ .

# Interior Point Methods (for LP)

## KKT conditions

$$\begin{aligned}
 c - A^\top \lambda - s &= 0 \\
 Ax &= b \\
 XSe &= \mu e \\
 x, s &\geq 0
 \end{aligned} \tag{KKT}_\mu$$

## Newton-Step

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} \xi_c \\ \xi_b \\ r_{xs} \end{bmatrix} := \begin{bmatrix} c - A^\top \lambda - s \\ b - Ax \\ \mu e - XSe \end{bmatrix}$$

# Interior Point Methods (for LP)

## KKT conditions

$$\begin{aligned}
 c - A^T \lambda - s &= 0 \\
 Ax &= b \\
 XSe &= \mu e \\
 x, s &\geq 0
 \end{aligned} \tag{KKT}_{\mu}$$

## Newton-Step

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} \xi_c \\ \xi_b \\ r_{xs} \end{bmatrix} := \begin{bmatrix} c - A^T \lambda - s \\ b - Ax \\ \mu e - XSe \end{bmatrix}$$

## Newton Step (reduced)

$$\begin{bmatrix} -\Theta & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \xi_c - X^{-1}r_{xs} \\ \xi_b \end{bmatrix}$$

where  $\Theta = X^{-1}S$ ,  $X = \text{diag}(x)$ ,  $S = \text{diag}(s)$

# Path Following Methods

- choose  $x_0, \lambda_0, s_0 > 0, \mu = x_0^\top s_0 / n$
- compute Newton step  $(\Delta x, \Delta s, \Delta \lambda)$  for (KKT) and given  $\mu^+ < \mu$ .
- compute stepsizes

$$\alpha = \max_{\alpha > 0} \{ \alpha : x + \alpha \Delta x \geq 0, s + \Delta s \geq 0, (x, s) \in \mathcal{N}_{2/-\infty}(\tau) \}$$

- take step

$$x_+ = x + 0.995 \alpha \Delta x$$

$$\lambda_+ = \lambda + 0.995 \alpha \Delta \lambda$$

$$s_+ = z + 0.995 \alpha \Delta s$$

- update  $\mu$ :

$$\mu_+ = \sigma \frac{x_+^\top s_+}{n}, \quad 0 < \sigma < 1$$

# Solving NLP by Interior Point Method

## NLP

$$\min f(x) \quad \text{s.t. } g(x) \leq 0 \quad (\text{NLP })$$

# Solving NLP by Interior Point Method

## NLP

$$\begin{aligned} \min f(x) - \mu \sum \ln z_i & \quad \text{s.t.} \quad g(x) + z &= 0 \\ & \quad z \geq 0 \end{aligned} \quad (\text{NLP}_\mu)$$

# Solving NLP by Interior Point Method

## NLP

$$\min f(x) - \mu \sum \ln z_i \quad \text{s.t.} \quad g(x) + z = 0 \quad (\text{NLP})$$

$$z \geq 0$$

## Optimality conditions

$$\begin{aligned}\nabla f(x) - A(x)^\top y &= 0 \\ g(x) + z &= 0 \\ XZ\mathbf{e} &= \mu e \\ x, z &\geq 0\end{aligned}$$

## Newton Step

$$\begin{bmatrix} Q(x, y) & A(x)^\top \\ A(x) & -\Theta \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \nabla - f(x) - A(x)^\top y \\ -g(x) - \mu Y^{-1} e \end{bmatrix}$$

where

$$\begin{aligned}Q(x, y) &= \nabla_{xx}^2(f(x) + y^\top g(x)), & A(x) &= \nabla g(x) \\ \Theta &= X^{-1}Z, & X &= \text{diag}(x), & Z &= \text{diag}(z)\end{aligned}$$

# Linear Algebra of IPMs

Main work: solve

$$\underbrace{\begin{bmatrix} -Q - \Theta & A^\top \\ A & 0 \end{bmatrix}}_{\Phi(QP)} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix} \quad \text{or}$$

$$\underbrace{\begin{bmatrix} -Q(x, y) & A(x)^\top \\ A(x) & \Theta \end{bmatrix}}_{\Phi(NLP)} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix}$$

$\Phi(QP)$

$\Phi(NLP)$

for several right-hand-sides at each iteration

Two stage solution procedure

- factorize  $\Phi = LDL^\top$
- backsolve(s) to compute direction  $(\Delta x, \Delta y)$  + corrections

$\Rightarrow \Phi$  changes numerically but not structurally at each iteration

**Key to efficient** implementation is exploiting structure of  $\Phi$  in these two steps

# Structure of matrices $A$ and $Q$ for SCOPF:

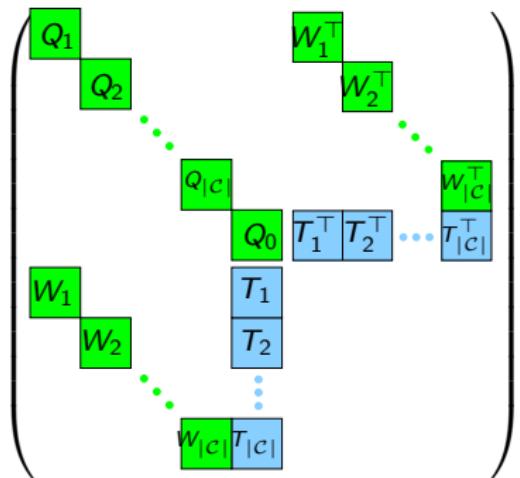
Matrix  $A(x)$

$$\left( \begin{array}{cc} W_1 & \\ & W_2 \\ & \ddots \\ & \ddots \\ & W_{|\mathcal{C}|} & T_{|\mathcal{C}|} \end{array} \right)$$

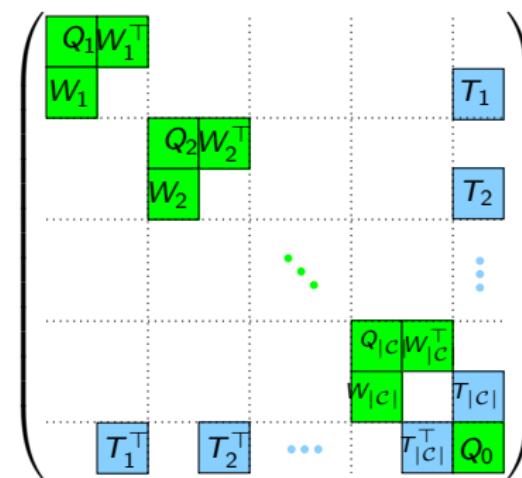
Matrix  $Q(x, y)$

$$\left( \begin{array}{cc} Q_1 & \\ & Q_2 \\ & \ddots \\ & \ddots \\ & Q_{|\mathcal{C}|} & Q_0 \end{array} \right)$$

# Structures of A and Q imply structure of $\Phi$ :



$$\begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix}$$



$$P \begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} P^{-1}$$

# Structures of A and Q imply structure of $\Phi$ :

$$\begin{pmatrix}
 Q_1 & W_1^\top & \\
 Q_2 & W_2^\top & \\
 \vdots & \ddots & \\
 Q_{|C|} & W_{|C|}^\top & \\
 Q_0 & T_1^\top & T_2^\top \\
 & T_1 & T_2 \\
 W_1 & & \\
 W_2 & & \\
 \vdots & & \\
 W_{|C|} & T_{|C|} & 
 \end{pmatrix}
 \quad
 \begin{pmatrix}
 \Phi_1 & B_1^\top & \\
 & \Phi_2 & B_2^\top \\
 & & \ddots \\
 & & \Phi_{|C|} & B_{|C|}^\top \\
 B_1 & B_2 & \cdots & B_{|C|} & \Phi_0
 \end{pmatrix}$$

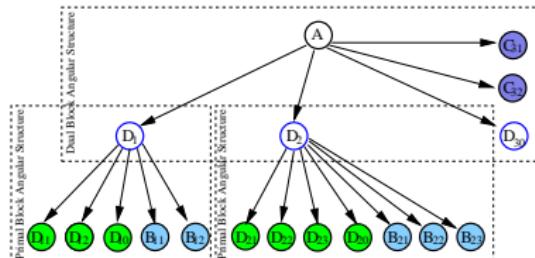
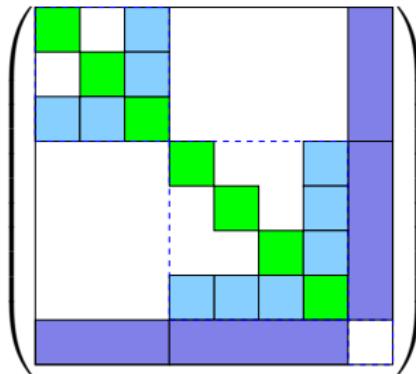
$$\begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} \quad P \begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} P^{-1}$$

Bordered block-diagonal structure in Augmented System!

# OOPS: Object Oriented Parallel Solver

## OOPS

- OOPS is an IPM implementation, that can exploit (nested) block structures through object oriented linear algebra
- Solved (multistage) stochastic programming problems from portfolio management with over  $10^9$  variables  
( $\approx 2\text{h}$  on 1280 processors)



# Exploiting Structure in IPM

## Block-Factorization of Augmented System Matrix

$$\underbrace{\begin{pmatrix} \Phi_1 & B_1^\top \\ \ddots & \vdots \\ & \Phi_n B_n^\top \\ B_1 \dots B_n & \Phi_0 \end{pmatrix}}_{\Phi} \underbrace{\begin{pmatrix} x_1 \\ \vdots \\ x_n \\ x_0 \end{pmatrix}}_x = \underbrace{\begin{pmatrix} b_1 \\ \vdots \\ b_n \\ b_0 \end{pmatrix}}_b$$

## Solution of Block-system by Schur-complement

The solution to  $\Phi x = b$  is

$$\begin{aligned} x_0 &= C^{-1} \tilde{b}_0, \quad \tilde{b}_0 = b_0 - \sum_i B_i \Phi_i^{-1} b_i \\ x_i &= \Phi_i^{-1} (b_i - B_i^\top x_0), \quad i = 1, \dots, n \end{aligned}$$

where  $C$  is the *Schur-complement*

$$C = \Phi_0 - \sum_{i=1}^n B_i \Phi_i^{-1} B_i^\top$$

$\Rightarrow$  only need to factor  $\Phi_i$ , not  $\Phi$

# Exploiting Structure in IPM

## Solution of Block-system by Schur-complement

The solution to  $\Phi x = b$  is

$$\begin{aligned}x_0 &= C^{-1} \tilde{b}_0, \quad \tilde{b}_0 = b_0 - \sum_i B_i \Phi_i^{-1} b_i \\x_i &= \Phi_i^{-1} (b_i - B_i^\top x_0), \quad i = 1, \dots, n\end{aligned}$$

where  $C$  is the *Schur-complement*

$$C = \Phi_0 - \sum_{i=1}^n B_i \Phi_i^{-1} B_i^\top$$

Bottlenecks in this process are

- Factorization of the  $\Phi_i$ ;
- Assembling  $\sum_{i=1}^n B_i \Phi_i^{-1} B_i^\top$

# Structure of Augmented System Matrix

$$\Phi = \begin{bmatrix} \Phi_1 & & & B_1^\top \\ & \Phi_2 & & B_2^\top \\ & & \ddots & \vdots \\ & & & \Phi_n & B_n^\top \\ B_1 & B_2 & \cdots & B_n & \Phi_0 \end{bmatrix}, \quad \Phi_i = \begin{bmatrix} D_i & W_i^T \\ W_i & 0 \end{bmatrix}$$

For DC-OPF

$$W_i = \begin{bmatrix} R & A_i^T \\ A_i & 0 \end{bmatrix}, \quad B_i^\top = \begin{bmatrix} 0 \\ J \end{bmatrix},$$

For AC-OPF

$$W_i = \begin{bmatrix} H_u^{f/t} & I \\ G_u^Q & J \\ G_u^P & \end{bmatrix}, \quad B_i^\top = \begin{bmatrix} H_{V_g}^{f/t} \\ G_{V_g}^Q \\ G_{V_g}^P & J \end{bmatrix}$$

# Block-Factorization for DC-OPF

## Structure of $\Phi_i$ for DC-OPF

$$\Phi_i = \begin{bmatrix} D_i & W_i^T \\ W_i & 0 \end{bmatrix}, \quad W_i = \begin{bmatrix} R & A_i^T \\ A_i & 0 \end{bmatrix}, \quad B_i^\top = \begin{bmatrix} 0 \\ J \end{bmatrix},$$

## Factorization of $\Phi_i$ :

$W_i$  is invertible and constant throughout IPM iterations

- To solve  $\Phi_i x = b$  only  $W_i$  needs to be factored:

$$\begin{bmatrix} D_i & W_i^T \\ W_i & 0 \end{bmatrix} \begin{bmatrix} x^{(0)} \\ x^{(1)} \end{bmatrix} = \begin{bmatrix} b^{(0)} \\ b^{(1)} \end{bmatrix}$$

$$\Rightarrow x^{(1)} = W_i^{-1} b^{(1)}, \quad x^{(0)} = D_i^{-1} (b^{(0)} - W_i^T x^{(1)})$$

## To build $B_i \Phi_i^{-1} B_i^\top$

$$\begin{aligned} B_i \Phi_i^{-1} B_i^\top &= -J^\top W_i^{-\top} D_i W_i^{-1} J \\ &= -V_i D_i V_i^\top, \quad V_i^\top = W_i^{-1} J \end{aligned}$$

# Exploiting Structure in IPM

Solution of Block-system by Schur-complement

The solution to  $\Phi x = b$  is

$$\begin{aligned}x_0 &= C^{-1} \tilde{b}_0, \quad \tilde{b}_0 = b_0 - \sum_i B_i \Phi_i^{-1} b_i \\C &= \Phi_0 + \sum_{i=1}^n V_i D_i V_i^\top, \quad V_i^\top = W_i^{-1} J\end{aligned}$$

Forming  $V_i D_i V_i^\top$  is expensive

# Exploiting Structure in IPM

## Solution of Block-system by Schur-complement

The solution to  $\Phi x = b$  is

$$\begin{aligned}x_0 &= C^{-1} \tilde{b}_0, \quad \tilde{b}_0 = b_0 - \sum_i B_i \Phi_i^{-1} b_i \\C &= \Phi_0 + \sum_{i=1}^n V_i D_i V_i^\top, \quad V_i^\top = W_i^{-1} J\end{aligned}$$

Forming  $V_i D_i V_i^\top$  is expensive

⇒ Solve  $Cx_0 = \tilde{b}_0$  by iterative method

- Use (preconditioned) iterative method (e.g. GMRES)
- with  $M = \Phi_0 + nV_0 D_0 V_0^\top$  as preconditioner for SCOPF  
(Qiu, Flueck '05)

⇒ Evaluating residuals  $r = \tilde{b}_0 - Cx_0$  is easy:

$$Cx_0 = \Phi_0 x_0 + \sum J^\top W_i^{-\top} D_i W_i^{-1} J x_0, \quad J \text{ is 0-1 matrix}$$

# State of the Art: Summary

## IPM for OPF: State of the Art

- Structure exploiting Linear Algebra within IPM
- Schur-complement approach with compact factorizations
- Preconditioned iterative solver for Schur-complement matrix
- All done automatically through OOPS.

Problems are still very large

Pan European network with 13000 nodes, 20000 lines

$\Rightarrow \approx 10^{10}$  variables/constraints for (n-1) SCOPF. (+ wind!)

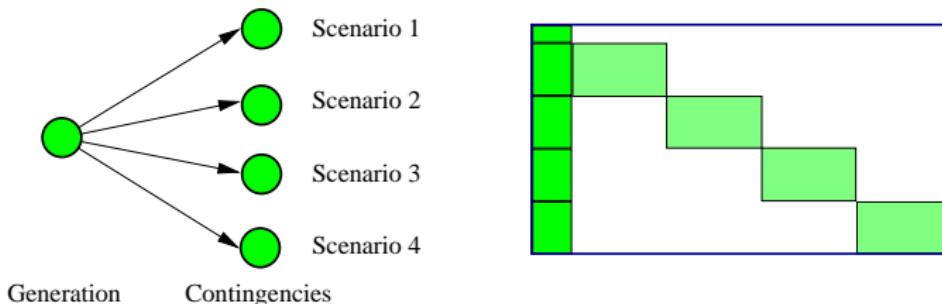
## Novel Approaches

- Structured IPM Crash-Start
- (Dynamic) Contingency/Scenario Generation

# Structured IPM Crash Start

## Idea

- SCOPF (like many other structured problems) consists of a **small core** that is **repeated** many times.  
⇒ First solve **much smaller** problem of **same structure**
- Use solution as **advanced starting point** for the full problem



## But

- IPMs are notoriously bad at exploiting a known starting point

# Warmstarting Interior Point Methods

Aim: Use information from solution process of

$$\begin{aligned} \min c^\top x \quad & \text{s.t. } Ax = b \\ & x \geq 0 \end{aligned} \tag{LP}$$

to construct a starting point for (nearby problem)

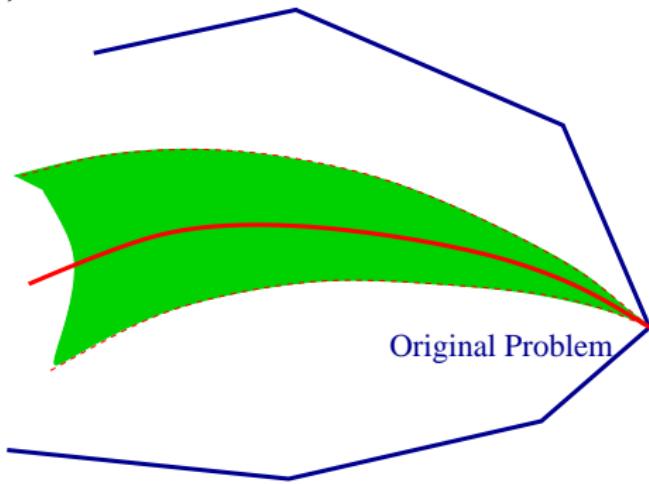
$$\begin{aligned} \min \tilde{c}^\top x \quad & \text{s.t. } \tilde{A}x = \tilde{b} \\ & x \geq 0 \end{aligned} \tag{\widetilde{LP}}$$

where  $\tilde{A} \approx A, \tilde{b} \approx b, \tilde{c} \approx c$

- It is **not** a good idea to use the solution of (LP) to start ( $\widetilde{LP}$ ).
- *Unlike for the Simplex/Active Set Method!*

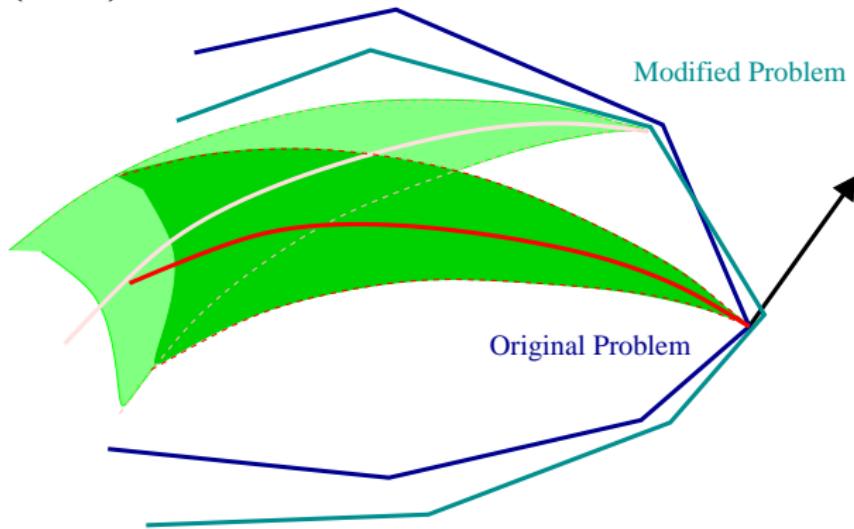
# Why?

Hipolito (1993): Search direction is parallel to nearby constraints



# Why?

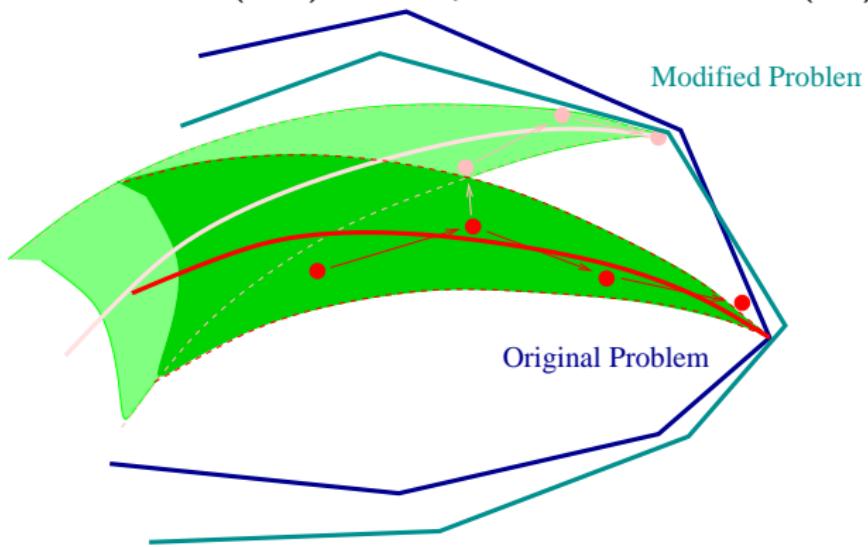
Hipolito (1993): Search direction is parallel to nearby constraints



⇒ only small step in search direction can be taken

# Warmstarting Heuristics

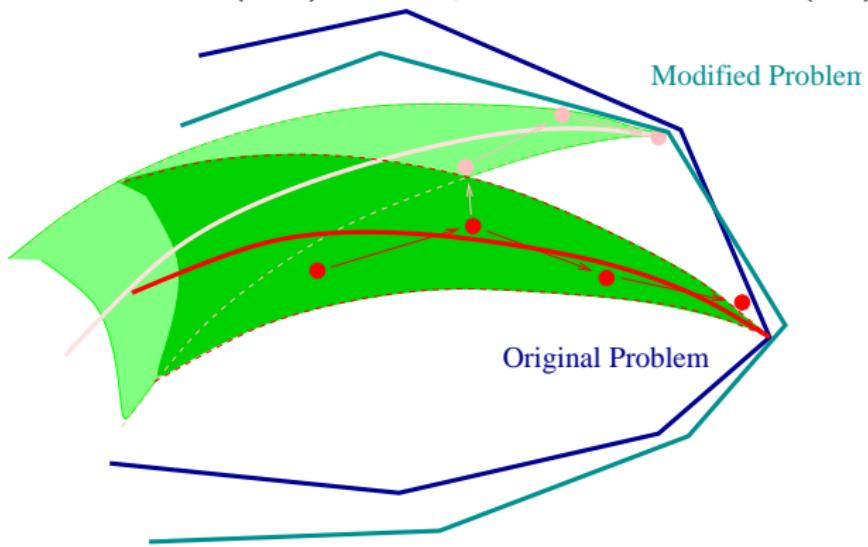
Idea: Start close to the (new) central path, not close to the (old) solution



⇒ Start from a previous iterate and do additional *modification* step.

# Warmstarting Heuristics

Idea: Start close to the (new) central path, not close to the (old) solution

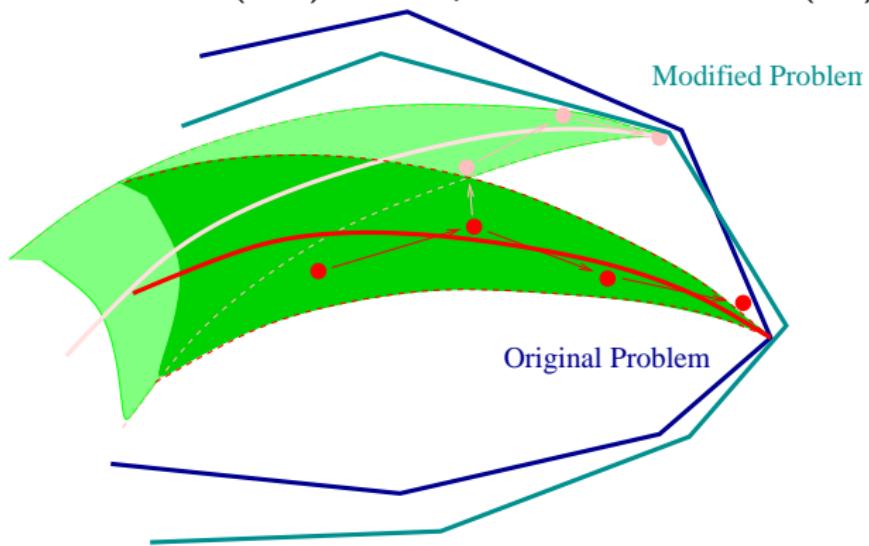


## IPM Warmstart: State-of-the-Art

- Can save (consistently) about 50%-60% of IPM iterations
- Across all problem sizes (up to  $\approx 10^8$  variables)

# Warmstarting Heuristics

Idea: Start close to the (new) central path, not close to the (old) solution



## IPM Crash-start

- Find (cheaply) a point near the central path of the problem (corresponding to an appropriate  $\mu$ -value)

# Interior Point Warmstarts: Theoretical Results

A typical warmstart result is (Assume  $\tilde{A} = A$ ):

**Lemma** (based on Yıldırım/Wright '02)

Let  $(x, \lambda, s) \in \mathcal{N}_2(\theta_0)$  for problem (LP) the the full mod. step  $(\Delta x, \Delta \lambda, \Delta s)$  in the perturbed problem ( $\tilde{L}\tilde{P}$ ) is feasible and

$$(x + \Delta x, \lambda + \Delta \lambda, s + \Delta s) \in \mathcal{N}_2(\theta)$$

provided that

$$\delta_{bc} \leq \frac{\theta - \theta_0}{2C(d)} \min \left\{ \frac{1}{2n+1}, \frac{\mu}{4C(d)\|d\|} \right\}$$

$\Rightarrow$  “small  $\delta_{bc}$ , large  $\mu$ ”

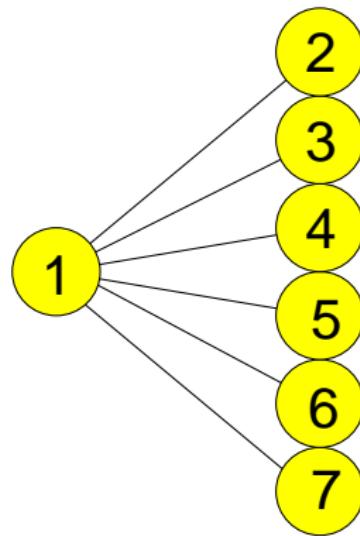
$C(d)$  is the Renegar condition number of the problem  $d = (A, b, c)$ :

$$C(d) = \frac{\|d\|}{\rho(d)}, \quad \rho(d) = \text{"distance to infeasibility"}$$

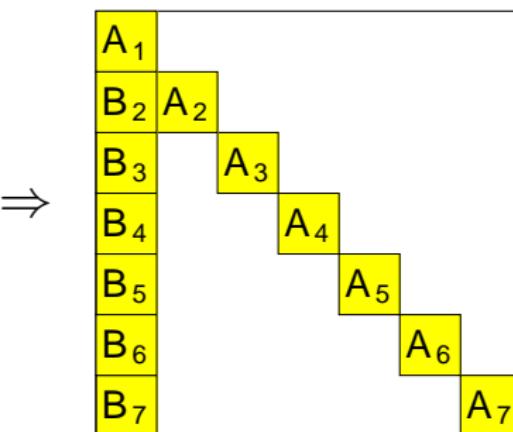
and

$$\delta_{bc} := \frac{\Delta c}{\|d\|} + 2C(d) \frac{\Delta b}{\|d\|}$$

# Structured IPM Crash-start



$C_1 \ C_2 \ C_3 \ C_4 \ C_5 \ C_6 \ C_7$

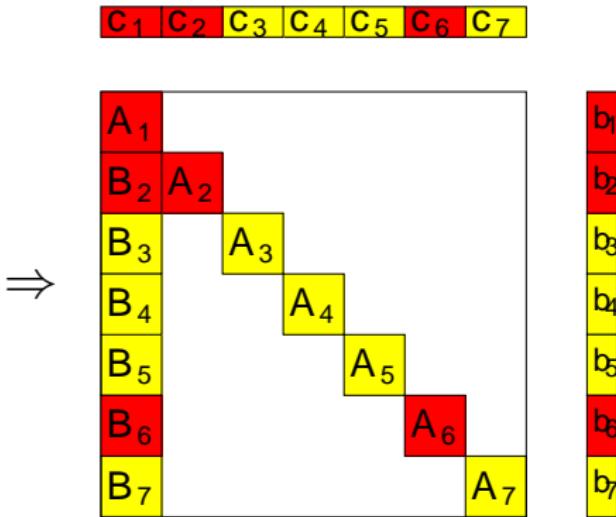
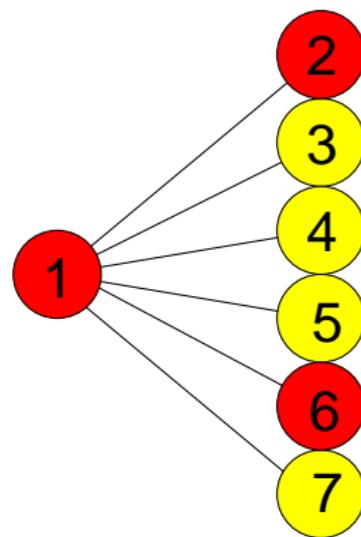


$b_1 \ b_2 \ b_3 \ b_4 \ b_5 \ b_6 \ b_7$

Idea:

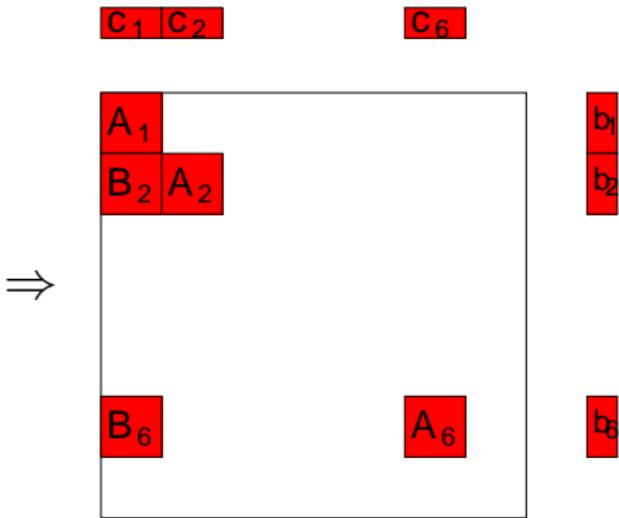
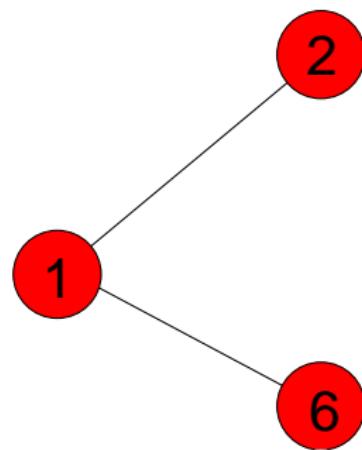
- Choose a few sample scenarios
- Generate a central point for the reduced problem
- And extend it to a warmstart point for the full problem

# Structured IPM Crash-start



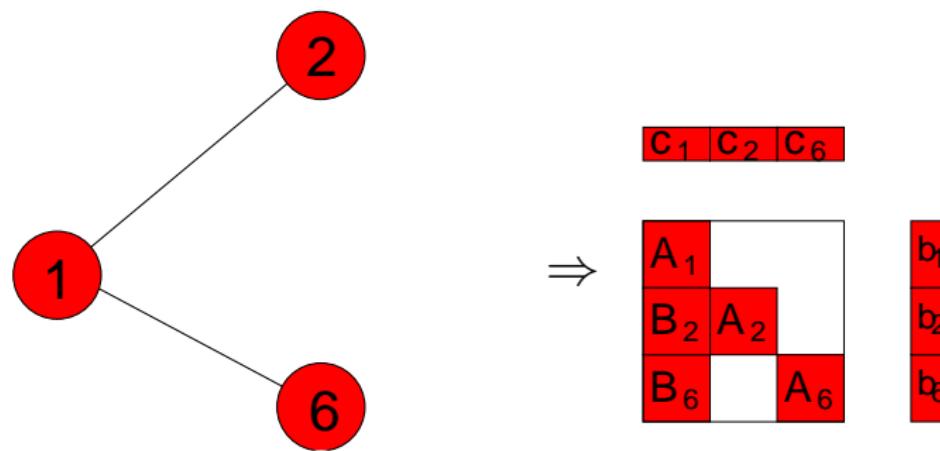
- Select sample scenarios

# Structured IPM Crash-start



- Select sample scenarios
- Reduce Problem

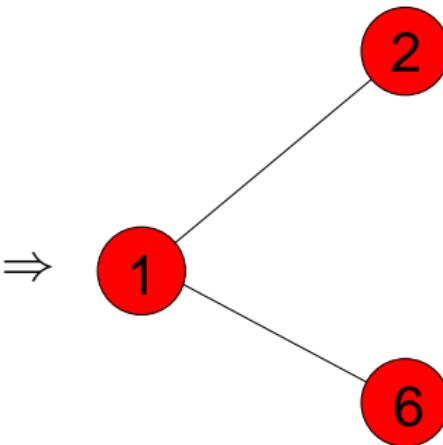
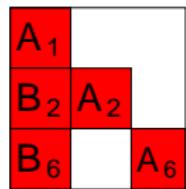
# Structured IPM Crash-start



- Select sample scenarios
- Reduce Problem

# Structured IPM Crash-start

$c_1 \mid c_2 \mid c_6$



To solve the problem by warmstarting, reverse the process

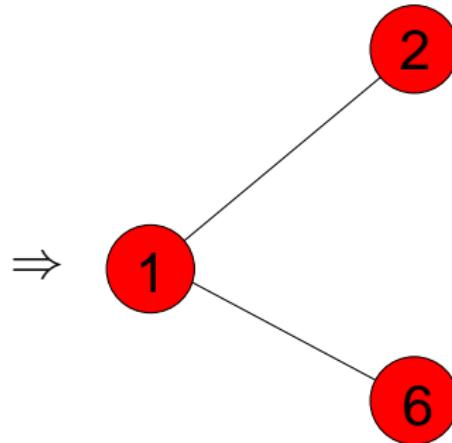
# Structured IPM Crash-start

C <sub>1</sub>	C <sub>2</sub>	C <sub>6</sub>
----------------	----------------	----------------

X <sub>1</sub>	X <sub>2</sub>	X <sub>6</sub>
Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>6</sub>

A <sub>1</sub>		
B <sub>2</sub>	A <sub>2</sub>	
B <sub>6</sub>		A <sub>6</sub>

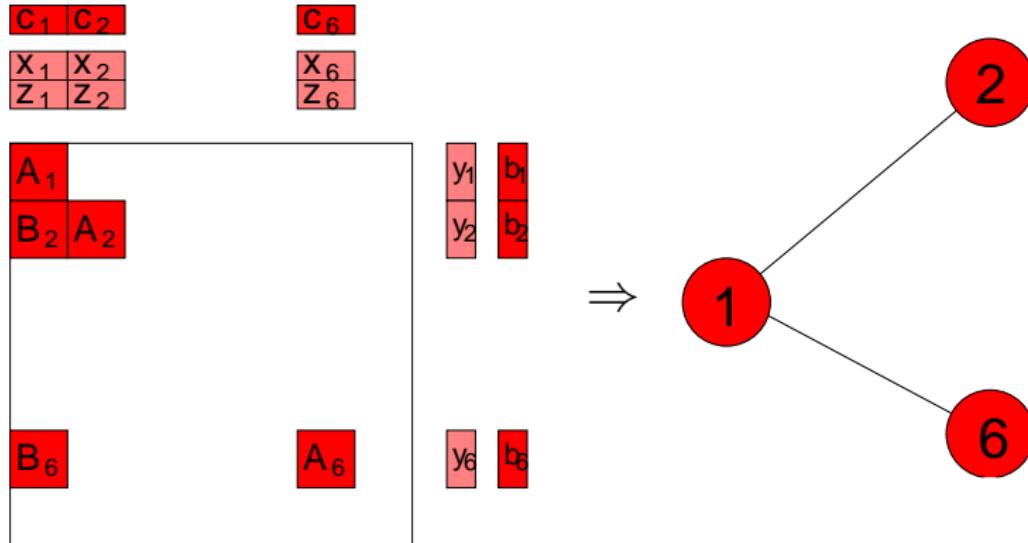
y <sub>1</sub>	b <sub>1</sub>
y <sub>2</sub>	b <sub>2</sub>
y <sub>6</sub>	b <sub>6</sub>



To solve the problem by warmstarting, reverse the process

- Find central point for reduced problem

# Structured IPM Crash-start



To solve the problem by warmstarting, reverse the process

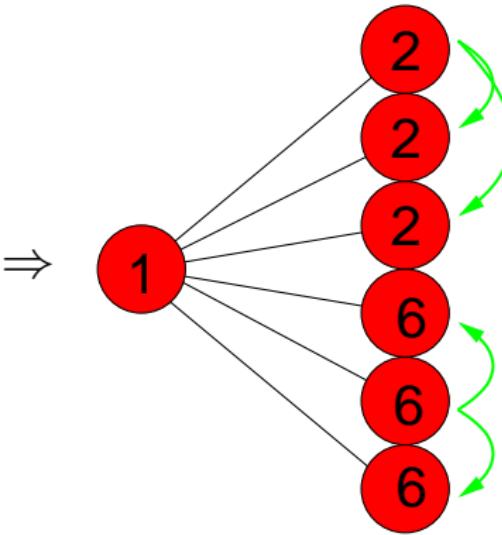
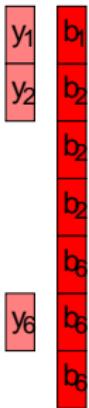
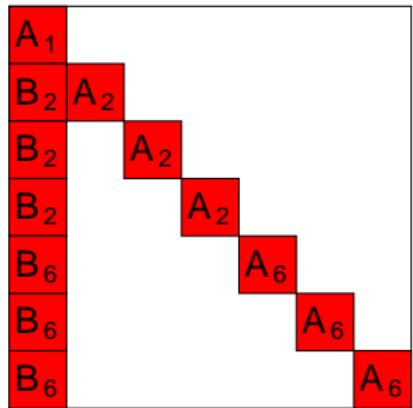
- Find central point for reduced problem
- Expand the problem to original size

# Structured IPM Crash-start

$C_1 \ C_2 \ C_2 \ C_2 \ C_6 \ C_6 \ C_6$

$X_1 \ X_2$   
 $Z_1 \ Z_2$

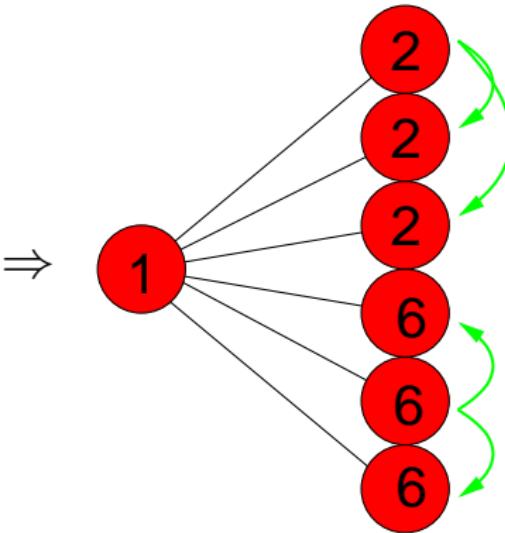
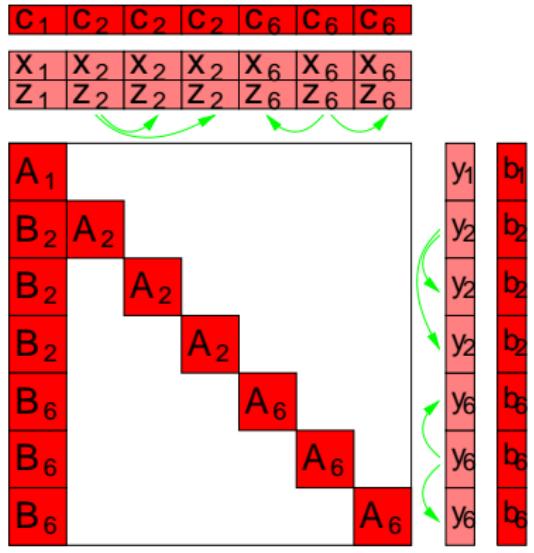
$X_6$   
 $Z_6$



To solve the problem by warmstarting, reverse the process

- Find central point for reduced problem
- Expand the problem to original size (by duplicating scenarios)

# Structured IPM Crash-start

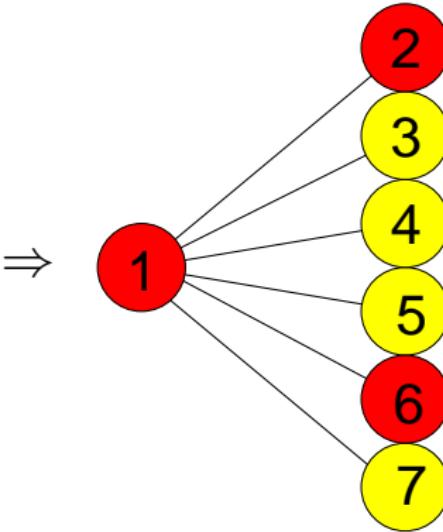
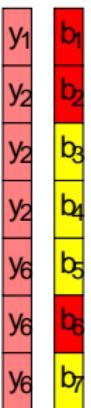
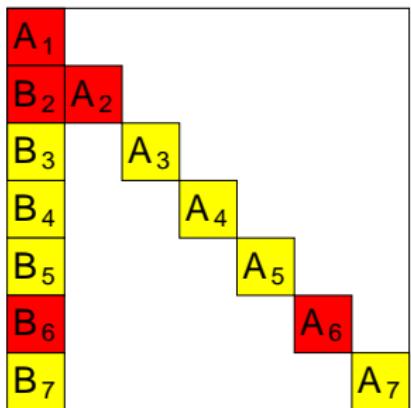


To solve the problem by warmstarting, reverse the process

- Find central point for reduced problem
- Expand the problem to original size (by duplicating scenarios)
- Expand solution to **primal/dual feasible** and **near central** point for expanded problem

# Structured IPM Crash-start

C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>
X <sub>1</sub>	X <sub>2</sub>	X <sub>2</sub>	X <sub>2</sub>	X <sub>6</sub>	X <sub>6</sub>	X <sub>6</sub>
Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>2</sub>	Z <sub>2</sub>	Z <sub>6</sub>	Z <sub>6</sub>	Z <sub>6</sub>

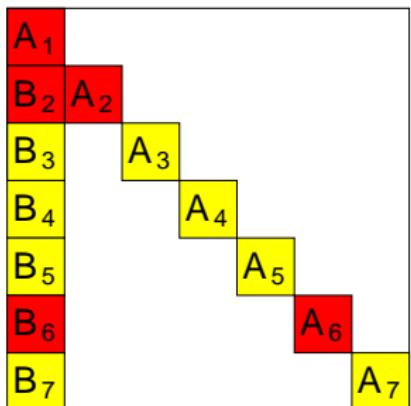


To solve the problem by warmstarting, reverse the process

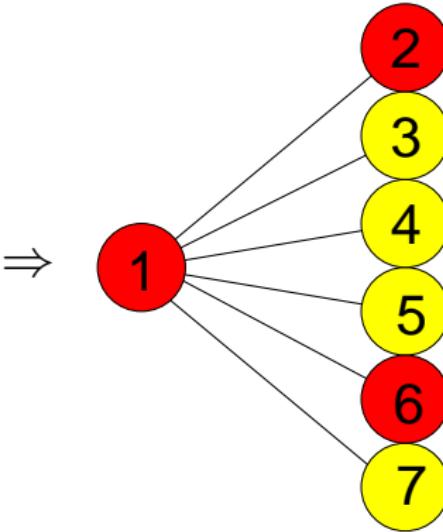
- Find central point for reduced problem
- Expand the problem to original size (by duplicating scenarios)
- Expand solution to primal/dual feasible and near central point for expanded problem
- Use this to warmstart full problem

# Structured IPM Crash-start

C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>
X <sub>1</sub>	X <sub>2</sub>	X <sub>2</sub>	X <sub>2</sub>	X <sub>6</sub>	X <sub>6</sub>	X <sub>6</sub>
Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>2</sub>	Z <sub>2</sub>	Z <sub>6</sub>	Z <sub>6</sub>	Z <sub>6</sub>



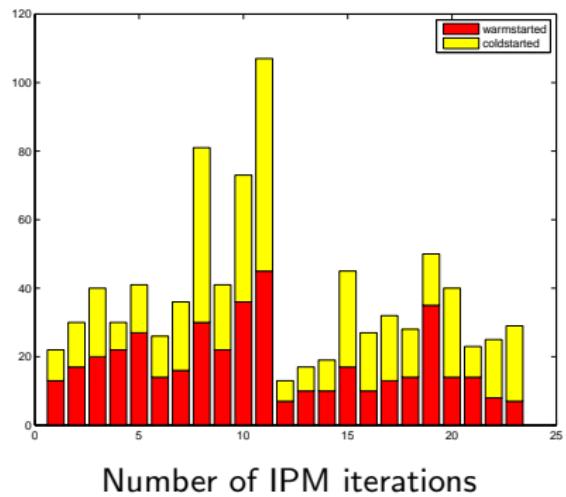
y <sub>1</sub>	b <sub>1</sub>
y <sub>2</sub>	b <sub>2</sub>
y <sub>2</sub>	b <sub>3</sub>
y <sub>2</sub>	b <sub>4</sub>
y <sub>6</sub>	b <sub>5</sub>
y <sub>6</sub>	b <sub>6</sub>
y <sub>6</sub>	b <sub>7</sub>



- Scheme has been implemented and analysed for Stochastic Programming
- Extension uses sequence of progressively larger problems that each approximate the next one in the sequence.  
(Colombo, G.: *Multilevel warmstart*)

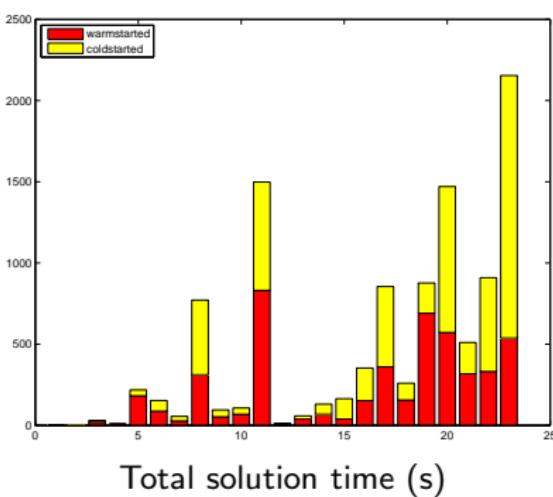
# Reduced Tree Warmstart: Results

- SP test problems & Capacity assignment problems
- Ranging from 1,000 - 100,000 variables



Number of IPM iterations

-50.5%



Total solution time (s)

-42.1%

→ Colombo, Gondzio, G. (2009)

# Multilevel Warmstart Results

## Stochastic Programming Testproblems

Problem	scenarios	cold	2-step	multistep
ex1	20000	580	327	302
	40000	1559	766	701
ex3	10000	563	346	316
	20000	1793	626	586
s97	10000	498	389	307
s98	10000	3189	826	481
j99	10000	1796	375	295
Minoux	10000	2644	1212	1176
JII_gva	4000	4981	2523	2251
T1B3	10000	995	663	637
r4c	10000	2098	835	749

(Solution time in seconds)

# Crash-Start Application: Contingency Generation

- “n-1”- (or even “n-2”-security) requires the inclusion of many contingency scenarios.
- Pan-European system has 13000 nodes and 20000 lines  
⇒ Resulting SCOPF model has  $\approx 10^{10}$  variables.
- Only a **few** contingencies are **critical** for operation of the system (but which ones)?

## Contingency Generation

- Generate contingency scenarios dynamically when needed
- For DC OPF feasibility constraints for each contingency can be derived explicitly, checked and included into the model (Berry, Dunnett '89).
- No generalisation for AC OPF (yet) - No equivalent cuts can be derived.

# Contingency Generation

## Prototype Algorithm:

Set up the model with a few base scenarios

Solve model to obtain power generation  $P^*$ .

**repeat**

    Check for violated contingency scenarios.

    Add violated scenarios to the model

    Re-solve model to obtain new power generation  $P^*$ .

**until** no more violated contingencies

# Contingency Generation

## Prototype Algorithm:

Set up the model with a few base scenarios

Solve model to obtain power generation  $P^*$ .

**repeat**

    Check for violated contingency scenarios.

    Add violated scenarios to the model

    Re-solve model to obtain new power generation  $P^*$ .

**until** no more violated contingencies

## Warmstart

- The above scheme results in a series of SCOPF models each with an increasing number of contingencies
- Can add single scenarios to warmstart point  
(Colombo, G.: Decomposition based warmstart)

# Contingency Generation

## Prototype Algorithm:

Set up the model with a few base scenarios.

Solve model to obtain power generation  $P^*$ .

**repeat**

    Check for violated contingency scenarios.

    Add violated scenarios to the model.

    Re-solve model to obtain new power generation  $P^*$ .

**until** no more violated contingencies

# Contingency Generation

Prototype Algorithm:

Set up the model with a few base scenarios.  $\mu_0 = \bar{\mu}, k = 0$ .

Solve model to obtain  $\mu_0$ -center.  $\Rightarrow P_{\mu_0}$ .

**repeat**

Check for violated contingency scenarios.

Add violated scenarios to the model.

Choose  $\mu_{k+1} : 0 < \mu_{k+1} < \mu_k, k \leftarrow k + 1$

Warmstart and iterate to find  $\mu_k$ -center.  $\Rightarrow P_{\mu_k}$ .

**until** no more violated contingencies

# Contingency Generation

## Prototype Algorithm:

Set up the model with a few base scenarios.  $\mu_0 = \bar{\mu}, k = 0$ .

Solve model to obtain  $\mu_0$ -center.  $\Rightarrow P_{\mu_0}$ .

**repeat**

Check for violated contingency scenarios.

**for all** violated scenarios **do**

Set up single scenario problem with  $P = P_{\mu_k}$  and solve for  
 $\mu_k$ -center

**end for**

Add violated scenarios to the model.

**Patch together warmstart point for expanded problem.**

Choose  $\mu_{k+1} : 0 < \mu_{k+1} < \mu_k, k \leftarrow k + 1$

Warmstart and iterate to find  $\mu_k$ -center.  $\Rightarrow P_{\mu_k}$ .

**until** no more violated contingencies

# Modelling Languages

## Bottleneck: Model generation

How to generate the model in a form that is understandable by a structure exploiting solver?

Traditionally this is done by handcrafted C/C++ code.

- Cumbersome/Error-prone
- Difficult to change model

# Modelling Languages

Bottleneck: Model generation

How to generate the model in a form that is understandable by a structure exploiting solver?

Traditionally this is done by handcrafted C/C++ code.

- Cumbersome/Error-prone
- Difficult to change model

⇒ use Modelling Language?

# Modelling Languages

## Bottleneck: Model generation

How to generate the model in a form that is understandable by a structure exploiting solver?

Traditionally this is done by handcrafted C/C++ code.

- Cumbersome/Error-prone
- Difficult to change model

⇒ use Modelling Language?

- Is aware of sparsity but not of structure
- Can not parallelise the model generation
- Often cannot generate entire problem on one processor

# Modelling Languages

## Bottleneck: Model generation

How to generate the model in a form that is understandable by a structure exploiting solver?

Traditionally this is done by handcrafted C/C++ code.

- Cumbersome/Error-prone
- Difficult to change model

⇒ use Modelling Language?

- Is aware of sparsity but not of structure
- Can not parallelise the model generation
- Often cannot generate entire problem on one processor

## S(P)ML!

Structure Conveying Parallelisable Modelling Language

# Structure Conveying Modelling Language: S(P)ML

S(P)ML is an extension to AMPL

- Mimic the “block” nature of the problem using block keyword:

```
block nameOfBlock{j in NODES}: {  
    ...  
}
```

- Blocks can contain sets, variables and constraints, even nested blocks.
- These elements are repeated over the indexing expression
- Scope of these elements delimited by block { ... }
- Reference variables outside their scope using object-oriented syntax:

```
nameOfBlock[j].nameOfElement
```

# DC-SCOPF model in S(P)ML

```
set BUSES; set GENERATORS;set LINES within (BUSES cross BUSES);
set CONTINGENCIES within LINES;
param V, react{LINES}, demand{BUSES};
var P{GENERATORS};

block Cont{k in CONTINGENCIES}:  {
    set REMLINES = LINES diff {k};
    var Flow{l in REMLINES} >=FlowLim[l] ,<=FlowLim[l]; delta{BUSES};
    subject to KCL{b in BUSES}:
        sum{g in GENERATORS:bus(g)==b} P[g]
        = sum{l in REMLINES:source(l)==b} Flow[l]+demand[b];
    subject to KVL{l in REMLINES}:
        Flow[l] = -V*V/react[l]*sum{b in BUSES:source(l)=b}delta[b];
}
minimize cost sum{g in GENERATORS} P[g]*c1[g] + P[g]*P[g]*c2[g];
```

# DC-SCOPF model in S(P)ML

```
set BUSES; set GENERATORS;set LINES within (BUSES cross BUSES);
set CONTINGENCIES within LINES;
param V, react{LINES}, demand{BUSES};
var P{GENERATORS};

block Cont{k in CONTINGENCIES}:  {
    set REMLINES = LINES diff {k};
    var Flow{l in REMLINES} >=FlowLim[l] ,<=FlowLim[l]; delta{BUSES};
    subject to KCL{b in BUSES}:
        sum{g in GENERATORS:bus(g)==b} P[g]
        = sum{l in REMLINES:source(l)==b} Flow[l]+demand[b];
    subject to KVL{l in REMLINES}:
        Flow[l] = -V*V/react[l]*sum{b in BUSES:source(l)=b}delta[b];
}
minimize cost sum{g in GENERATORS} P[g]*c1[g] + P[g]*P[g]*c2[g];
```

- Clear separation of model into **global (linking)** part

# DC-SCOPF model in S(P)ML

```
set BUSES; set GENERATORS;set LINES within (BUSES cross BUSES);
set CONTINGENCIES within LINES;
param V, react{LINES}, demand{BUSES};
var P{GENERATORS};

block Cont{k in CONTINGENCIES}:  {
    set REMLINES = LINES diff {k};
    var Flow{l in REMLINES} >=FlowLim[l] ,<=FlowLim[l]; delta{BUSES};

    subject to KCL{b in BUSES}:
        sum{g in GENERATORS:bus(g)==b} P[g]
        = sum{l in REMLINES:source(l)==b} Flow[l]+demand[b];

    subject to KVL{l in REMLINES}:
        Flow[l] = -V*V/react[l]*sum{b in BUSES:source(l)=b}delta[b];
}

minimize cost sum{g in GENERATORS} P[g]*c1[g] + P[g]*P[g]*c2[g];
```

- Clear separation of model into global (linking) part
- and local (repeated) parts

# DC-SCOPF model in S(P)ML

```

set BUSES; set GENERATORS;set LINES within (BUSES cross BUSES);
set CONTINGENCIES within LINES;
param V, react{LINES}, demand{BUSES};
var P{GENERATORS};

block Cont{k in CONTINGENCIES}:  {
    set REMLINES = LINES diff {k};
    var Flow{l in REMLINES} >=FlowLim[l] ,<=FlowLim[l]; delta{BUSES};

    subject to KCL{b in BUSES}:
        sum{g in GENERATORS:bus(g)==b} P[g]
        = sum{l in REMLINES:source(l)==b} Flow[l]+demand[b];

    subject to KVL{l in REMLINES}:
        Flow[l] = -V*V/react[l]*sum{b in BUSES:source(l)=b}delta[b];
}

minimize cost sum{g in GENERATORS} P[g]*c1[g] + P[g]*P[g]*c2[g];

```

- Clear separation of model into global (linking) part
- and local (repeated) parts
- Can be analysed by preprocessor

# DC-SCOPF model in S(P)ML

```
set BUSES; set GENERATORS;set LINES within (BUSES cross BUSES);
set CONTINGENCIES within LINES;
param V, react{LINES}, demand{BUSES};
var P{GENERATORS};

block Cont{k in CONTINGENCIES}:  {
    set REMLINES = LINES diff {k};
    var Flow{l in REMLINES} >=FlowLim[l] ,<=FlowLim[l]; delta{BUSES};

    subject to KCL{b in BUSES}:
        sum{g in GENERATORS:bus(g)==b} P[g]
        = sum{l in REMLINES:source(l)==b} Flow[l]+demand[b];

    subject to KVL{l in REMLINES}:
        Flow[l] = -V*V/react[l]*sum{b in BUSES:source(l)=b}delta[b];
}

minimize cost sum{g in GENERATORS} P[g]*c1[g] + P[g]*P[g]*c2[g];
```

- Clear separation of model into global (linking) part
- and local (repeated) parts
- Can be analysed by preprocessor
- Provide construct for (multistage) stochastic programming

# DC-SCOPF model in S(P)ML

```
set BUSES; set GENERATORS;set LINES within (BUSES cross BUSES);
set CONTINGENCIES within LINES;
param V, react{LINES}, demand{BUSES};
var P{GENERATORS};

block Cont{k in CONTINGENCIES}:  {
    set REMLINES = LINES diff {k};
    var Flow{l in REMLINES} >=FlowLim[l] ,<=FlowLim[l]; delta{BUSES};

    subject to KCL{b in BUSES}:
        sum{g in GENERATORS:bus(g)==b} P[g]
        = sum{l in REMLINES:source(l)==b} Flow[l]+demand[b];

    subject to KVL{l in REMLINES}:
        Flow[l] = -V*V/react[l]*sum{b in BUSES:source(l)=b}delta[b];
}

minimize cost sum{g in GENERATORS} P[g]*c1[g] + P[g]*P[g]*c2[g];
```

- Can be downloaded from [www.maths.ed.ac.uk/ERGO](http://www.maths.ed.ac.uk/ERGO)

# Conclusions & Further Work

## Conclusions

- IPM can solve structured OPF problems efficiently.
- Not just DC, also AC
- Not just ( $n-1$ ) secure, also probabilistic models
- Warmstart and contingency generation offer potential for further speed-up
- Can be parallelised efficiently
- SPML offers appropriate modelling environment

## Further Work

- Contingency generation is a “naive multilevel algorithm”. Can we turn it into a proper multilevel method?
- How to test for violated contingencies efficiently?